Introduction

H. Irem Türkmen

irem@yildiz.edu.tr

Yıldız Technical University

Computer Engineering Department

BLM5106- Advanced Algorithm Analysis and Design

Algorithm Analysis

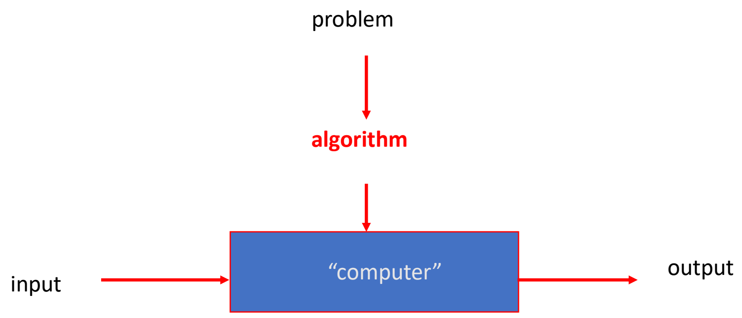
- Resources :
  - Introduction to Design & Analysis of Algorithms, Anany Levitin, 2011
  - Cormen, Leiserson, Rivest, Stein, "Introduction to Algorithms, 3E", MIT Press, 2009
  - Algorithms, Fourth Edition, R. Sedgewick and K. Wayne (http://algs4.cs.princeton.edu), 2013
  - The Algorithm Design Manual, Steven Skiena,2010
- Grading :
  - Project: 1
  - Midterm Exam(s) 1
  - Final Exam

Contents

Contents

Notion of Algorithm

- An algorithm is a sequence of unambiguous instructions for solving a problem, for obtaining a required output for any legitime input in a finite amount of time.



Analyzing an Algorithm

- How good is the algorithm?
  - Correctness
  - Efficiency (Time, Space) An algorithm is efficient if it has a polynomial running time.
  - Simplicity
  - Generality

- Does there exist a better algorithm?
  - Lower bounds
  - Optimality

Fundamental Data Structures

- **Linear Data Structures**
  - Arrays, Linked lists, Stack, Queue
- **Graphs**
  - Nodes, Edges, Adjacency matrix and Adjacency lists
- **Trees**
  - Rooted Trees, Ordered Trees
- **Sets and Dictionaries**
  - *Universal set, Bit vector,* Using the list structure

Fundamentals of the Analysis of Algorithm Efficiency

- System independent effects :
  - Algorithm
- System dependent effects
  - Hardware : CPU, memory, cache
  - Software : compiler, interpreter, etc.
  - System : OS, network, etc.

Fundamentals of the Analysis of Algorithm Efficiency

- Measuring an Input's Size
  - Product of two matrices: total number of elements *N* in the matrices being multiplied
  - Evaluating a polinomial: polynomial's degree
- Units for Measuring Running Time

$$T(n) \approx c_{op} C(n)$$

Assuming that *C(n)* = 1/2 *n(n* − 1*)*, how much longer will the algorithm run if we double its input size?

$$C(n) = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \approx \frac{1}{2}n^2$$

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4.$$

Orders of growth

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	**constant**	`a = b + c;`	statement	add two numbers	1
$\log N$	**logarithmic**	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search	~ 1
N	**linear**	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum	2
$N \log N$	**linearithmic**	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	**quadratic**	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs	4
N^3	**cubic**	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples	8
2^N	**exponential**	[see combinatorial search lecture]	exhaustive search	check all subsets	

Worst-Case, Best-Case, and Average-Case Efficiencies

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

//Searches for a given value in a given array by sequential search
//Input: An array $A[0..n-1]$ and a search key K
//Output: The index of the first element in A that matches K
//   or -1 if there are no matching elements
$i \leftarrow 0$
while $i < n$ **and** $A[i] \neq K$ **do**
    $i \leftarrow i + 1$
if $i < n$ **return** i
else return -1

- ***Worst-case*** $C_{worst}(n) = n$
- ***Best-case*** $C_{best}(n) = 1$
- ***Average-case*** $C_{avg}(n) = [1*p/n+2*p/n+...+i*p/n+...+n*p/n]+n(1-p)$

(a) probability of a successful search is equal to p ($0 \leq p \leq 1$)

(b) probability of the first match occurring in the ith position of the list is the same for every i